CADUCEUS: PARALLEL SPECULATIVE DECODING WITH TWO SIMPLE DRAFTERS INTERTWINED

Anonymous authors
Paper under double-blind review

ABSTRACT

Speculative decoding can reduce target-model passes by verifying multiple draft tokens at once, but its end-to-end speedup depends on balancing proposal quality against drafting and verification cost. Small language models offer high-quality drafts at the cost of latency, whereas n-gram retrieval is fast but depends on close matches. We introduce Caduceus, a training-free speculative decoding framework that combines these complementary drafters around a persistent draft tree. Within the compound drafter, n-gram retrieval and small-model expansion alternate: retrieval adds candidate paths to the persistent tree, and the small model scores and extends the tree to produce new anchors for subsequent retrieval. Target observations also update a prompt-local online n-gram store. At runtime, the drafter continues growing the tree while the target model verifies a snapshot, and compatible branches persist across verification rounds. Across four target models from 8B to 70B, Caduceus achieves $2.13\times$–$3.70\times$ speedup on Spec-Bench and $2.27\times$–$3.86\times$ on HumanEval over target-only autoregressive decoding, attaining the highest full-benchmark throughput among the baselines reported for each of the eight benchmark–model settings. On repeated-sampling SpecTTS-Bench with a Llama-70B target, Caduceus achieves $3.99\times$ speedup on the first rollout and $4.39\times$ on the third.

1 INTRODUCTION

Autoregressive decoding remains a bottleneck in large language model serving because tokens are generated sequentially. In low-batch inference, each step produces only one token while accessing the target model's parameters, so decoding is often constrained by memory bandwidth rather than arithmetic throughput (Leviathan et al., 2023; Chen et al., 2023; Cai et al., 2024). Speculative decoding shifts part of this work to a cheaper drafter, which proposes a sequence or tree of candidate tokens for parallel verification by the target model. When several candidates are accepted, one target-model forward pass commits multiple output tokens while preserving the target model's output distribution (Leviathan et al., 2023; Chen et al., 2023; Miao et al., 2024).

Fewer target verification passes, however, do not automatically yield lower wall-clock latency. End-to-end speedup depends jointly on how many tokens each target verification pass commits, how long the draft takes to construct, and how much work is required to verify it. A larger draft model or a longer and wider draft can improve candidate coverage, but it also adds autoregressive drafting or verification work; a cheaper but inaccurate drafter leaves much of that work unproductive (Leviathan et al., 2023; Yan et al., 2025; Zhao et al., 2024; Chen et al., 2024). Effective speculative decoding must therefore balance proposal quality against both drafting and verification cost.

Existing drafters make different quality–efficiency trade-offs. Off-the-shelf small language models produce high-quality proposals across varied inputs, but their autoregressive generation leaves a non-negligible drafting cost. N-gram retrieval can copy long continuations quickly, but depends on finding a reliable match (He et al., 2024; Oliaro et al., 2025). Target-specific learned drafters can improve proposal quality or reduce sequential drafting cost, but require specialized training and model-specific draft components (Cai et al., 2024; Li et al., 2025a; 2024; 2025b; Huang et al., 2025; Chen et al., 2026; Hu et al., 2026; Shi et al., 2026).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Finding an efficient drafter is one part of reducing end-to-end latency. Another is to hide drafting time by allowing drafting to proceed concurrently with target verification (McDanel et al., 2025; Liu et al., 2025; Zhang et al., 2025).

We present Caduceus, named for two forms of intertwining around a persistent draft tree. Within the compound drafter, n-gram retrieval alternates with small-model expansion; at the system level, compound drafting overlaps with target verification. Retrieval adds candidate paths to the persistent tree, and the small model scores and extends the tree to produce new anchors for subsequent retrieval. Target verification commits a path and prunes incompatible branches, while compatible branches remain available for subsequent growth.

This organization also lets Caduceus inherit the n-gram drafter's online adaptation: target observations update a prompt-local n-gram store across repeated rollouts. On SpecTTS-Bench, the speedup of Caduceus generally rises as this store accumulates target observations.

Across four target models, Caduceus achieves $2.13\times$–$3.70\times$ speedup on Spec-Bench and $2.27\times$–$3.86\times$ on HumanEval over target-only autoregressive decoding. It attains the highest full-benchmark throughput among the baselines reported for each of the eight benchmark–model settings, with domain-level speedup reaching $6.33\times$ on Spec-Bench. Our contributions are:

- A training-free compound drafter that alternates n-gram retrieval with small-model expansion: retrieval accelerates drafting, while the small model controls token quality and selects roots for the next retrieval step.
- A three-level n-gram hierarchy with two static count-based stores—global L1 and optional domain-specific L2—and an online, prob-based prompt-local L3 store.
- An asynchronous runtime centered on the persistent draft tree: the compound drafter grows the tree while target verification selects a path and prunes incompatible branches.

2 Related Work

Training-free drafting. Classical speculative decoding uses an off-the-shelf smaller autoregressive model (Leviathan et al., 2023; Chen et al., 2023). Other training-free approaches use prompt lookup, corpus or suffix matching, adaptive n-gram statistics, dense retrieval, and recycled target candidates (Saxena, 2023; He et al., 2024; Oliaro et al., 2025; Ou et al., 2024; Chen et al., 2025a; Gritta et al., 2025; Luo et al., 2025). Token Recycling builds a draft tree from target-generated candidates stored in an online adjacency matrix (Luo et al., 2025). RACER maintains an LRU-managed Aho–Corasick automaton for online pattern retrieval and fills the remaining per-step draft budget with candidates derived from copied target logits (Zhang et al., 2026).

Trained drafting. Medusa adds multiple trained decoding heads for parallel candidate generation, whereas the EAGLE family trains target-feature-conditioned draft modules; EAGLE-2 further adapts the draft-tree shape using context-dependent draft confidence (Cai et al., 2024; Li et al., 2025a; 2024; 2025b). FastEagle, DFlash, JetSpec, and SpecBlock reduce sequential drafting with a trained cascade, block diffusion, a causal-parallel tree head, or block-iterative drafters, respectively (Huang et al., 2025; Chen et al., 2026; Hu et al., 2026; Shi et al., 2026). These methods all require a trained draft component adapted to the target model before deployment and are therefore not usable out of the box.

Hybrid drafting. Staged Speculative Decoding and CS-Drafting arrange lightweight and neural predictors in successive speculative stages, while Ouroboros reduces autoregressive drafting through phrase-level candidates (Spector & Ré, 2023; Chen et al., 2025b; Zhao et al., 2024). Hybrid Verified Decoding (HVD) predicts the payoff of a cache draft and chooses between cache verification and a model-based drafter, whereas Graft attaches retrieved paths to positions opened by pruning a model-generated tree (Su et al., 2026; Shen et al., 2026). Caduceus instead repeatedly alternates n-gram retrieval with small-model expansion within one persistent tree, using the small model to score branches from both sources and select subsequent retrieval anchors.

Asynchronous draft–target execution. PipeSpec arranges multiple models in a hierarchical producer–consumer pipeline with asynchronous coordination and rollback to reduce inter-stage idle

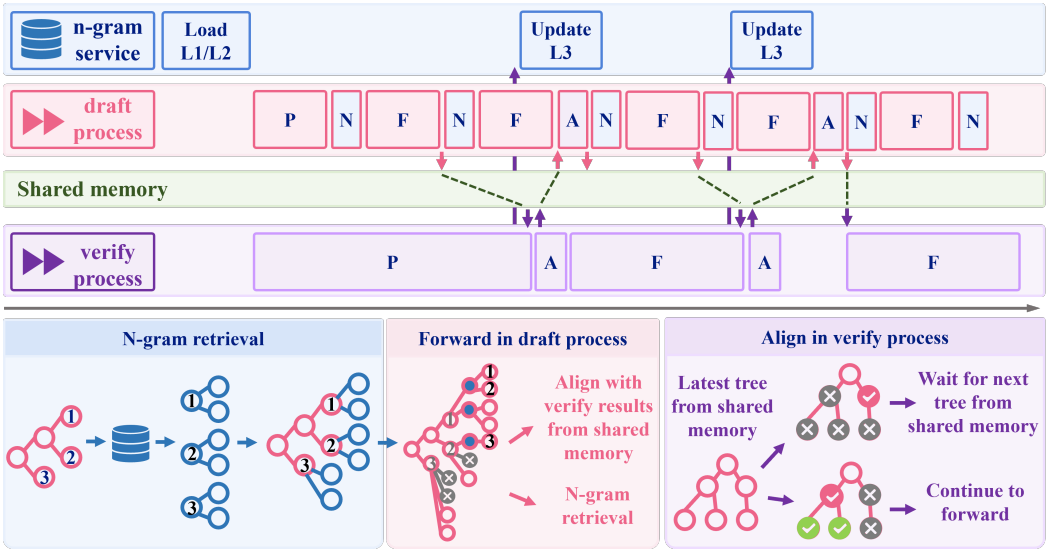


Figure 1: **Caduceus pipeline.** The three processes are the n-gram service (P0), draft process (P1), and verify process (P2). Shared memory carries tree snapshots and alignment requests between processes. Timeline markers P, N, A, and F denote prefill, n-gram retrieval, alignment, and model forward pass, respectively.

time. PEARL uses preverify and postverify to overlap drafting with target verification and adapt the number of drafted tokens. SwiftSpec combines asynchronous disaggregation with parallel tree generation, tree-aware KV-cache management, and fused kernels to reduce drafting and communication overhead (McDanel et al., 2025; Liu et al., 2025; Zhang et al., 2025). Caduceus instead centers asynchronous execution on a persistent draft tree: the compound drafter continues growing while target verification selects a path and prunes incompatible branches.

3 Method

3.1 Overview

Let p and q denote the target model and an off-the-shelf small model, respectively. Caduceus coordinates three concurrent processes around a persistent draft tree $\mathcal{T}$: the n-gram service (P0) provides CPU-side retrieval, the draft process (P1) runs q and grows $\mathcal{T}$, and the verify process (P2) runs p to verify drafts and commit output tokens. Within P1, retrieval and small-model expansion alternate; across P1 and P2, drafting overlaps with target-model verification. Figure 1 illustrates this execution; the complete procedure appears as Algorithm 1 in Appendix A.

3.2 Alternating N-gram and Small-Model Drafting

P1 avoids rebuilding an independent proposal in every round by maintaining a persistent draft tree $\mathcal{T}$ across target verification passes. Each update alternates n-gram retrieval and small-model expansion: P1 selects the k_a highest-scoring leaves as expansion anchors, queries P0 at these anchors, attaches the returned retrieval paths to $\mathcal{T}$, runs a forward pass of q with ancestor-only attention, and retains the top k_q children per expanded node under q to grow $\mathcal{T}$.

Let h denote the prompt concatenated with the currently committed output. For a candidate node v, let $y_{1:d}(v)$ denote its d-token path from h. Using ancestor-only attention, P1 scores v along this root-to-node path:

$$s_q(v \mid h) = \sum_{i=1}^{d} \log q(y_i \mid h \,\|\, y_{<i}), \tag{1}$$

Thus, $s_q(v \mid h)$ is the cumulative log probability that q assigns to the path ending at v.

3.3 Heterogeneous Three-Level N-gram Retrieval

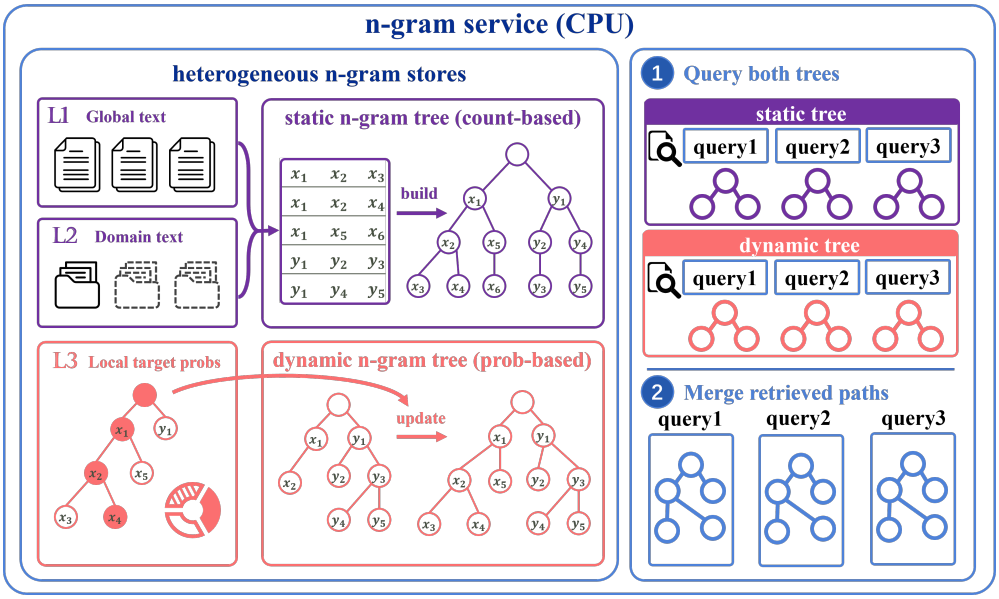


Figure 2: **Heterogeneous three-level n-gram retrieval.** Global L1 and optional domain-specific L2 jointly populate a single static count-based n-gram tree, whereas prompt-local L3 uses a dynamic prob-based n-gram tree updated from target distributions. P0 merges their candidate paths into a retrieval tree.

P0 organizes n-gram retrieval into three levels with different scopes and statistics (Figure 2), drawing on suffix retrieval and reuse of target-model predictions (Oliaro et al., 2025; Luo et al., 2025). Global L1 and optional domain-specific L2 jointly populate a single static count-based n-gram tree, while prompt-local L3 forms a temporary prob-based n-gram tree. Both trees map a suffix s to sparse next-token entries $(a, r(a \mid s))$, where a is a candidate next token and $r(a \mid s)$ is its edge weight. L1/L2 ingest offline token sequences. Let $C(s)$ be the number of occurrences of s and $C(s{\to}a)$ the number immediately followed by a; the static-tree weight is

$$r_{\text{stat}}(a \mid s) = \frac{C(s{\to}a)}{C(s)}, \qquad C(s) > 0. \tag{2}$$

L3 instead ingests target-model distributions computed during prefill and verification, retaining the top K next-token entries from each distribution. For an evaluated prefix h_t, let $\widetilde{p}_t(a \mid h_t)$ equal $p(a \mid h_t)$ for its top-K tokens and zero otherwise. Each observation updates every suffix s of h_t up to the maximum n-gram depth. If s has received n_s observations, L3 updates its sparse weights by

$$r_{\text{dyn}}^{(n_s+1)}(\cdot \mid s) = \text{TopK}_K\left(\frac{n_s r_{\text{dyn}}^{(n_s)}(\cdot \mid s) + \widetilde{p}_t(\cdot \mid h_t)}{n_s + 1}\right), \tag{3}$$

where TopK_K retains the K largest entries. Prompt transitions initialize L3 as one-hot observations; prefill and subsequent verification passes add target-distribution observations. L1/L2 can draw on

large offline corpora that typically contain only token sequences, making counts a practical statistic. L3 has fewer prompt-specific observations, so retaining target probabilities uses the information in each observation more fully.

P0 queries both trees by longest-suffix matching and merges their retrieved paths by token prefix into one retrieval tree. The n-gram retrieval budget M limits the merged tree returned for each expansion anchor. Appendix A.3 gives the store composition and retrieval procedure.

3.4 Asynchronous Drafting–Verification Pipeline

While P2 performs target-model verification, P1 continues growing $\mathcal{T}$ and publishing snapshots S, each bounded by the live-tree budget of H draft nodes. After each verification pass, P2 sends an alignment request to P1 and immediately aligns the latest snapshot locally to the updated context h. Local alignment retains the snapshot subtree compatible with h. If alignment succeeds and the retained subtree contains at least $\min(\lambda, B)$ draft nodes, P2 starts the next verification pass without waiting for P1; here λ is the lookahead threshold and B is the verification budget. Otherwise, P2 waits for P1's alignment and a sufficiently large aligned snapshot.

Upon receiving the request, P1 aligns $\mathcal{T}$ and its KV cache to h, removes incompatible branches, and rebases surviving nodes' path scores to the new context. If no compatible descendants remain, P1 resumes growth from h.

P1 and P2 have different tree-size requirements. A larger $\mathcal{T}$ retains more potentially reusable descendants across alignments, whereas a large verification input can substantially increase target-model forward latency. P2 therefore selects an ancestor-closed verification subtree D of at most B draft nodes from the aligned snapshot S; the separately supplied committed root token is excluded from B. Using the path weights $w_q(v \mid h) = \exp s_q(v \mid h)$ and viewing trees as their node sets, the selection objective is

$$D \in \arg \max_{\substack{D' \subseteq S,\ |D'| \le B \\ D' \text{ is ancestor-closed in } S}} \sum_{v \in D'} w_q(v \mid h). \tag{4}$$

In practice, we sort nodes by $s_q(v \mid h)$ in descending order and take the top B, breaking ties in favor of ancestors. Because a child's cumulative path score cannot exceed its parent's, the selected nodes are ancestor-closed. Appendix A.2 gives the derivation, and Appendix A.4 analyzes the pipeline's critical path.

3.5 Verification Rule

P2 evaluates all nodes in D in one target-model forward pass using the same ancestor-only attention rule as P1. At each distinct verified prefix, P2 applies the target decoding rule ρ once—argmax for greedy decoding or a draw from the target distribution for stochastic decoding—to select the next token. The selected tokens form a single connected output path from the committed root, which P2 commits. Because ancestor-only attention reproduces the autoregressive target distribution at every prefix and ρ is applied exactly once to each distinct prefix, the resulting output distribution is identical to that of autoregressive decoding.

4 Experiments

4.1 Experimental Setup

Benchmarks and protocol. We evaluate end-to-end throughput on three benchmarks: Spec-Bench (Xia et al., 2024), which contains 480 prompts spanning six generation domains; HumanEval (Chen et al., 2021), which contains 164 code-generation prompts; and SpecTTS-Bench (Sun et al., 2025), which contains 120 prompts and samples four solutions per prompt at temperature 0.6. The reported throughput experiments use a 512-token output cap; decoding mode and batch size are specified for each experiment. Model and n-gram datastore loading are completed before timing, while model prefill and token generation are included in the timed interval. Throughput is the arithmetic mean of per-prompt tokens per second (TPS), and speedup is the ratio to the corresponding

target-only autoregressive (AR) throughput. We use τ for the number of output tokens committed per target verification pass.

Models, hardware, and deployment. We evaluate four draft–target configurations: Llama-3.2-1B-Instruct→Llama-3.1-8B-Instruct, Qwen2.5-7B-Instruct→Qwen2.5-32B-Instruct-AWQ, Vicuna-7B→Vicuna-33B-AWQ, and Llama-3.2-1B-Instruct→Llama-3.3-70B-Instruct-AWQ. All measurements are performed on the same server equipped with NVIDIA A100 PCIe 40GB GPUs without NVLink. Llama-3.1-8B-Instruct, Qwen2.5-32B-Instruct-AWQ, and Vicuna-33B-AWQ each use one GPU for the target model, while Llama-3.3-70B-Instruct-AWQ is sharded across two GPUs. N-gram retrieval and its datastores run on the CPU. For every method with an independent draft model, the draft model is deployed on an additional GPU. Because EAGLE's trained draft head consumes target-model hidden states, placing it on a separate GPU would require additional cross-GPU transfers. We therefore follow the official implementation and colocate the head with the target model's final layer.

Caduceus configuration. Unless otherwise specified, Caduceus uses global L1 and prompt-local L3 retrieval. We disable optional L2 so that the main comparison does not rely on benchmark- or domain-matched offline corpora, and evaluate L2 separately on a held-out domain in Appendix B.3. We fix $K = 8$ for L3, $\lambda = 1$, $H = 448$, $M = 20$, $k_q = 5$, and $k_a = 3$. For the four model pairs listed above, B is set to 64, 32, 24, and 24, respectively.

Baselines. All baseline throughputs are measured in our evaluation environment. For baselines with tunable inference-time hyperparameters, we select configurations on a fixed 48-prompt subset before full-benchmark evaluation. PLD, SuffixDecoding, and Token Recycling are training-free, target-only baselines: the first two retrieve n-gram candidates, whereas Token Recycling reuses target-generated candidates (Saxena, 2023; Oliaro et al., 2025; Luo et al., 2025). Hugging Face assisted generation represents conventional autoregressive small-model drafting (Leviathan et al., 2023; Chen et al., 2023). We evaluate CS-Drafting using its released linear cascade; the Tree Attention extension reported in the paper is not included in the released implementation (Chen et al., 2025b). We further include PipeSpec and PEARL for overlapped draft–target execution (McDanel et al., 2025; Liu et al., 2025), and EAGLE-2 and EAGLE-3 for target-specific trained drafting (Li et al., 2024; 2025b).

4.2 Main Throughput Results

We compare the complete Caduceus system against all baselines on Spec-Bench and HumanEval under greedy decoding with batch size 1. In addition, we include three ablations to isolate the contributions of the two proposal sources and asynchronous execution: Caduceus-Ngram uses L1 and L3 without the small model or draft–verify overlap; Caduceus-SM keeps only the small-model proposal source and disables all n-gram sources; and Caduceus-Serial retains the compound drafter but disables draft–verify overlap.

Across the four target models, Caduceus achieves 2.13–$3.70\times$ speedup on Spec-Bench and 2.27–$3.86\times$ on HumanEval over target-only AR decoding (Figures 3 and 4; exact values are reported in Appendix B.1). It attains the highest reported throughput in all eight benchmark–model settings. With the 70B target, Caduceus reaches $6.33\times$ speedup on mathematical reasoning, $1.79\times$ that of EAGLE-3.

The 1B→8B pair is the least favorable setting for Caduceus. Target verification is comparatively fast and the draft–target latency gap is smaller, so small-model work is harder to hide and millisecond-scale tree-management and coordination costs are harder to amortize (Appendix A.4). Lightweight retrieval is therefore especially competitive in this setting: on Spec-Bench, Token Recycling reaches $1.82\times$, compared with $1.57\times$ for Caduceus-Ngram. Token Recycling benefits from its fixed tree structure, which allows tree masks and traversal indices to be precomputed. However, its released code provides only an 80-node template and cannot select a smaller tree budget for larger targets; its speedup falls to 1.08–$1.25\times$ on the 32B–70B targets.

The complete system outperforms Caduceus-Serial in all eight settings, supporting the benefit of draft–verify overlap. It also consistently outperforms Caduceus-Ngram, showing the benefit of compound drafting over retrieval-only drafting.

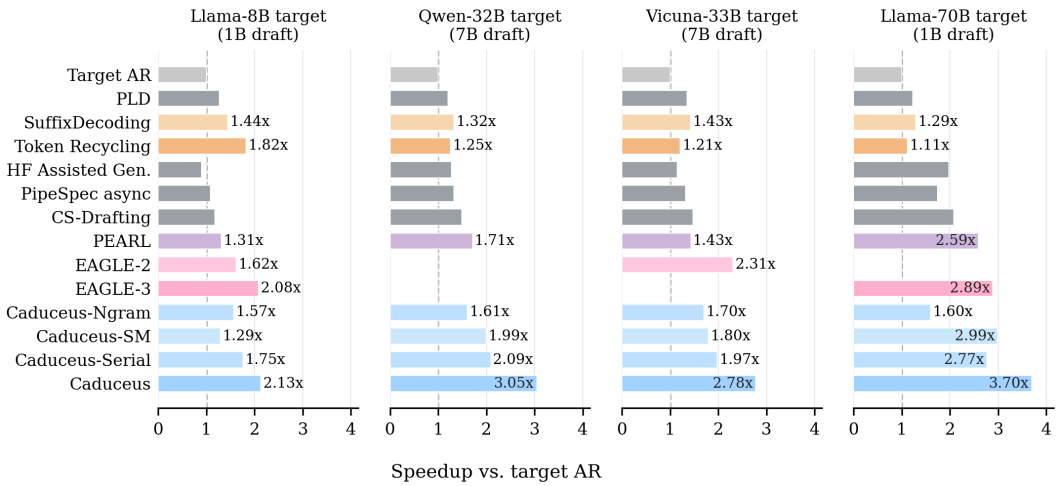


Figure 3: Spec-Bench throughput speedup over target-only AR decoding.

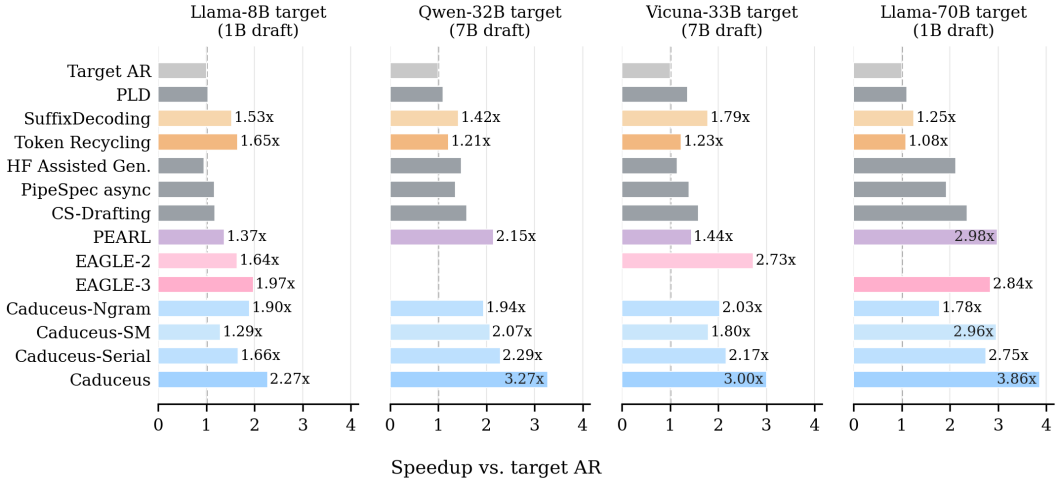


Figure 4: HumanEval throughput speedup over target-only AR decoding.

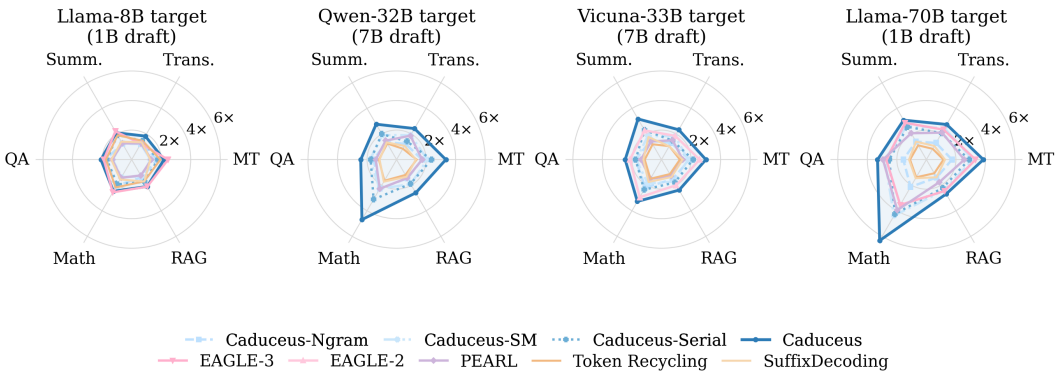


Figure 5: Spec-Bench throughput speedup over target-only AR decoding by domain. Domains are multi-turn chat (MT), translation (Trans.), summarization (Summ.), question answering (QA), mathematical reasoning (Math), and retrieval-augmented generation (RAG); all panels share the same radial scale.

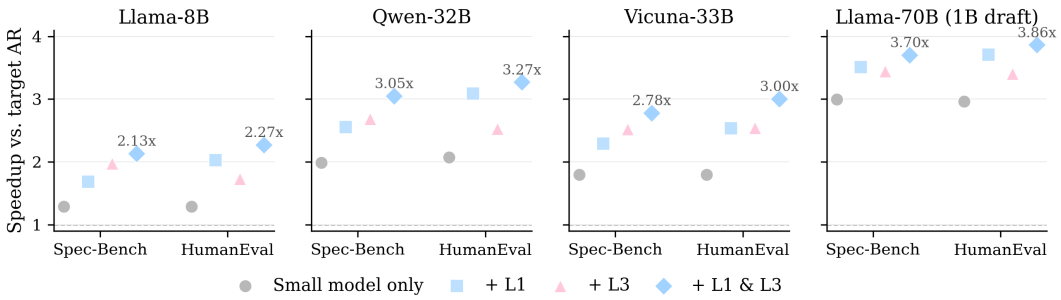


Figure 6: Speedup over target-only AR decoding when L1 and L3 are enabled independently and jointly.

4.3 Complementarity of L1 and L3

We isolate the two n-gram sources with a 2×2 ablation that independently enables global L1 and prompt-local L3 while keeping the small-model drafter and the remaining decoding configuration fixed. The four conditions are therefore small-model-only drafting, small model with L1, small model with L3, and the complete L1+L3 compound drafter.

Figure 6 shows that enabling either L1 or L3 improves over small-model-only drafting in every setting, while combining them achieves the highest throughput for all four targets on both Spec-Bench and HumanEval. Their relative contributions vary across model pairs and benchmarks: L1 supplies reusable patterns collected offline across requests, whereas L3 adapts to target-model observations from the current request. The consistent advantage of L1+L3 over either source alone supports the two sources as complementary components of the compound drafter.

4.4 Stochastic Decoding Results

We evaluate Llama 1B→8B and Llama 1B→70B and compare target-only AR, EAGLE-3, Caduceus-Ngram, and the complete Caduceus system. We retain L3 across the four rollouts of each prompt and clear it before moving to the next prompt. Caduceus-Ngram uses L1 and L3 without the small model or draft–verify overlap.

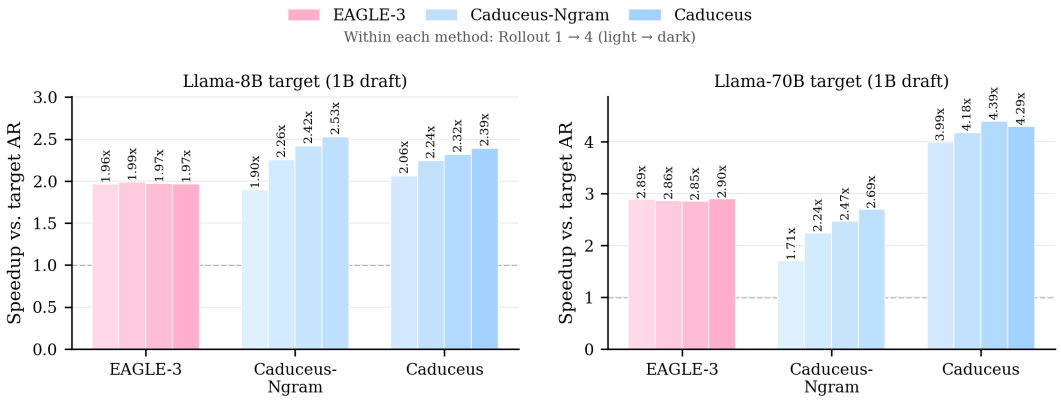


Figure 7: SpecTTS-Bench speedup under repeated sampling. Each method is evaluated on four rollouts per prompt; bars progress from the first to the fourth rollout from light to dark.

Caduceus reaches $2.25\times$ overall speedup for Llama 1B→8B and $4.21\times$ for Llama 1B→70B. On the 70B target, its speedup rises from $3.99\times$ on the first rollout to $4.39\times$ on the third, whereas EAGLE-

3 remains around 2.85–$2.90\times$. Caduceus-Ngram also improves across rollouts, consistent with L3 benefiting from accumulated target observations.

4.5 Batch-Size Scaling

We evaluate fixed batches on the full Spec-Bench with the Llama 1B→70B setup. A completed request leaves its slot idle until all requests in the current batch finish; no new request is admitted during a batch. Algorithm 2 in Appendix A summarizes the batched scheduler. Request TPS is the arithmetic mean of each request's generated tokens divided by its completion time, while aggregate batch TPS divides all generated tokens by total batch wall-clock time.

Table 1: Request and aggregate throughput across batch sizes.

	Request TPS			Aggregate batch TPS		
Batch size	AR	Caduceus	Speedup	AR	Caduceus	Speedup
2	6.084	21.272	$3.50\times$	10.027	34.558	$3.45\times$
4	5.691	14.232	$2.50\times$	15.879	43.884	$2.76\times$
8	5.242	9.920	$1.89\times$	26.728	58.696	$2.20\times$

Caduceus remains faster than batched AR at every tested batch size. As batch size increases from 2 to 8, its request-TPS speedup decreases from $3.50\times$ to $1.89\times$, while its aggregate-TPS speedup decreases from $3.45\times$ to $2.20\times$. Larger batches increase the arithmetic intensity and GPU utilization of batched AR, leaving less headroom for speculative decoding.

5 Conclusion

We presented Caduceus, a training-free speculative decoding framework that alternates n-gram retrieval with small-model expansion in a persistent draft tree and overlaps this compound drafter with target verification. Across four target models, Caduceus achieves $2.13\times$–$3.70\times$ speedup on Spec-Bench and $2.27\times$–$3.86\times$ on HumanEval over target-only AR decoding, attaining the highest full-benchmark throughput among the baselines reported for all eight benchmark–model settings. Ablations attribute these gains to compound drafting, complementary global and prompt-local retrieval, and draft–verify overlap. Caduceus also retains throughput gains under synchronized batching and stochastic repeated sampling, with the prompt-local L3 store benefiting from target observations accumulated across rollouts. These results show that complementary draft sources can be composed and scheduled asynchronously to improve speculative-decoding throughput without target-specific training.

6 Limitations

The non-model portions of the runtime include snapshot alignment, verification-buffer construction and transfer, posterior and acceptance processing, KV-cache compaction, verification-subtree selection, and inter-process coordination. Together, these operations add millisecond-scale latency to each verification step and remain to be optimized. Our stochastic-decoding experiments use a simple target-first verification rule; more acceptance-efficient verification under sampling remains future work. Finally, all experiments were conducted on NVIDIA A100 PCIe 40GB GPUs without NVLink; performance on other accelerators and interconnects remains unevaluated.

AI use statement

We used generative AI tools to assist with manuscript drafting, editing, and translation; to discuss methodological and experimental choices; to review mathematical derivations and interpret results; and to help write and debug analysis code and figures. We treated AI-generated material as suggestions, checked it against the implementation and experiment records, and take responsibility for the final text, code, figures, and claims.

Ethics statement

All experiments use publicly available benchmarks and text corpora, together with responses generated by the evaluated target models; no new human-subject data were collected. Because retrieval-based drafting may reproduce fragments of indexed material, deployments should ensure that indexed corpora comply with applicable licenses and data-use requirements.

Reproducibility statement

To support reproducibility, Section 3.5 specifies the verification rule, and Section 4.1 reports the evaluated models, hardware, decoding and timing protocols, and Caduceus configurations. Appendix A provides single-request and batched pseudocode together with details of tree attention, verification-subtree selection, and n-gram store construction and retrieval. Appendix B.1 reports absolute throughput, speedups, and committed-token statistics, while Appendix B.4 documents lookahead sensitivity and Appendix C reports system-level latency measurements.

References

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. Medusa: Simple LLM inference acceleration framework with multiple decoding heads, 2024. URL https://arxiv.org/abs/2401.10774.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023. URL https://arxiv.org/abs/2302.01318.

Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding, 2026. URL https://arxiv.org/abs/2602.06036.

Jinglin Chen, Qiwei Li, Zuchao Li, Baoyuan Qi, Guoming Liu, Haojun Ai, Hai Zhao, and Ping Wang. Faster in-context learning for LLMs via n-gram trie speculative decoding. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 18040–18051, Suzhou, China, November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.911. URL https://aclanthology.org/2025.emnlp-main.911/.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code, 2021. URL https://arxiv.org/abs/2107.03374.

Zhuoming Chen, Avner May, Ruslan Svirschevski, Yuhsun Huang, Max Ryabinin, Zhihao Jia, and Beidi Chen. Sequoia: Scalable and robust speculative decoding. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 129531–129563. Curran Associates, Inc., 2024. doi: 10.52202/079017-4116. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/ea1f5f0878d43ff4fb8bf64ef4a2326c-Paper-Conference.pdf.

Ziyi Chen, Xiaocong Yang, Jiacheng Lin, Chenkai Sun, Kevin Chen-Chuan Chang, and Jie Huang. Cascade speculative drafting for even faster LLM inference, 2025b. URL https://arxiv.org/abs/2312.11462.

Milan Gritta, Huiyin Xue, and Gerasimos Lampouras. DReSD: Dense retrieval for speculative decoding, 2025. URL https://arxiv.org/abs/2502.15572.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D. Lee, and Di He. REST: Retrieval-based speculative decoding, 2024. URL https://arxiv.org/abs/2311.08252.

Lanxiang Hu, Zhaoxiang Feng, Yulun Wu, Haoran Yuan, Yujie Zhao, Yu-Yang Qian, Bojun Wang, Peng Zhao, Daxin Jiang, Yibo Zhu, Tajana Rosing, and Hao Zhang. JetSpec: Breaking the scaling ceiling of speculative decoding with parallel tree drafting, 2026. URL https://arxiv.org/abs/2606.18394.

Haiduo Huang, Jiangcheng Song, Wenzhe Zhao, and Pengju Ren. FastEagle: Cascaded drafting for accelerating speculative decoding, 2025. URL https://arxiv.org/abs/2509.20416.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding, 2023. URL https://arxiv.org/abs/2211.17192.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-2: Faster inference of language models with dynamic draft trees, 2024. URL https://arxiv.org/abs/2406.16858.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE: Speculative sampling requires rethinking feature uncertainty, 2025a. URL https://arxiv.org/abs/2401.15077.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. EAGLE-3: Scaling up inference acceleration of large language models via training-time test, 2025b. URL https://arxiv.org/abs/2503.01840.

Tianyu Liu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, Winston Hu, and Xiao Sun. PEARL: Parallel speculative decoding with adaptive draft length, 2025. URL https://arxiv.org/abs/2408.11850.

Xianzhen Luo, Yixuan Wang, Qingfu Zhu, Zhiming Zhang, Xuanyu Zhang, Qing Yang, and Dongliang Xu. Turning trash into treasure: Accelerating inference of large language models with token recycling, 2025. URL https://arxiv.org/abs/2408.08696.

Bradley McDanel, Sai Qian Zhang, Yunhai Hu, and Zining Liu. PipeSpec: Breaking stage dependencies in hierarchical LLM decoding, 2025. URL https://arxiv.org/abs/2505.01572.

Xupeng Miao, Gabriele Oliaro, Zhihao Zhang, Xinhao Cheng, Zeyu Wang, Zhengxin Zhang, Rae Ying Yee Wong, Alan Zhu, Lijie Yang, Xiaoxiang Shi, Chunan Shi, Zhuoming Chen, Daiyaan Arfeen, Reyna Abhyankar, and Zhihao Jia. SpecInfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ASPLOS '24, pp. 932–949, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703867. doi: 10.1145/3620666.3651335. URL https://doi.org/10.1145/3620666.3651335.

Gabriele Oliaro, Zhihao Jia, Daniel Campos, and Aurick Qiao. SuffixDecoding: Extreme speculative decoding for emerging AI applications, 2025. URL https://arxiv.org/abs/2411.04975.

Jie Ou, Yueming Chen, and Wenhong Tian. Lossless acceleration of large language model via adaptive n-gram parallel decoding, 2024. URL https://arxiv.org/abs/2404.08698.

Apoorv Saxena. Prompt lookup decoding, November 2023. URL https://github.com/apoorvumang/prompt-lookup-decoding/.

Yuhao Shen, Tianyu Liu, Xinyi Hu, Quan Kong, Baolin Zhang, Jun Dai, Jun Zhang, Shuang Ge, Lei Chen, Yue Li, Mingcheng Wan, and Cong Wang. Draft less, retrieve more: Hybrid tree construction for speculative decoding, 2026. URL https://arxiv.org/abs/2605.20104.

Weijie Shi, Qiang Xu, Fan Deng, Yaguang Wu, Jiarun Liu, Yehong Xu, Hao Chen, Jia Zhu, Jiajie Xu, Xiangjun Huang, Jian Yang, and Xiaofang Zhou. SpecBlock: Block-iterative speculative decoding with dynamic tree drafting, 2026. URL https://arxiv.org/abs/2605.07243.

Benjamin Spector and Chris Ré. Accelerating LLM inference with staged speculative decoding, 2023. URL https://arxiv.org/abs/2308.04623.

Xin Su, Dawid Majchrowski, Fangyuan Yu, Vanshil Atul Shah, Sebastian Rogawski, Pawel Morkisz, Anahita Bhiwandiwalla, and Phillip Howard. Hybrid verified decoding: Learning to allocate verification in speculative decoding. *arXiv preprint arXiv:2606.01019*, 2026. URL https://arxiv.org/abs/2606.01019.

Shengyin Sun, Yiming Li, Xing Li, Yingzhao Lian, Weizhe Lin, Hui-Ling Zhen, Zhiyuan Yang, Chen Chen, Xianzhi Yu, Mingxuan Yuan, and Chen Ma. Scaling up, speeding up: A benchmark of speculative decoding for efficient LLM test-time scaling, 2025. URL https://arxiv.org/abs/2509.04474.

Heming Xia, Zhe Yang, Qingxiu Dong, Peiyi Wang, Yongqi Li, Tao Ge, Tianyu Liu, Wenjie Li, and Zhifang Sui. Unlocking efficiency in large language model inference: A comprehensive survey of speculative decoding, 2024. URL https://arxiv.org/abs/2401.07851.

Minghao Yan, Saurabh Agarwal, and Shivaram Venkataraman. Decoding speculative decoding, 2025. URL https://arxiv.org/abs/2402.01528.

Zihong Zhang, Zuchao Li, Lefei Zhang, Ping Wang, and Hai Zhao. RACER: Retrieval-augmented contextual rapid speculative decoding, 2026. URL https://arxiv.org/abs/2604.14885.

Ziyi Zhang, Ziheng Jiang, Chengquan Jiang, Menghan Yu, Size Zheng, Haibin Lin, Henry Hoffmann, and Xin Liu. SwiftSpec: Ultra-low latency LLM decoding by scaling asynchronous speculative decoding, 2025. URL https://arxiv.org/abs/2506.11309.

Weilin Zhao, Yuxiang Huang, Xu Han, Wang Xu, Chaojun Xiao, Xinrong Zhang, Yewei Fang, Kaihuo Zhang, Zhiyuan Liu, and Maosong Sun. Ouroboros: Generating longer drafts phrase by phrase for faster speculative decoding, 2024. URL https://arxiv.org/abs/2402.13720.

A Additional Method Details

Algorithm 1 Caduceus (single request).

Inputs: prompt x; target model p; off-the-shelf small model q; decoding rule ρ; static n-gram stores $\mathcal{L}_1$ and optional $\mathcal{L}_2$

Parameters: k_a: expansion anchors per update; k_q: children per expanded node; M: n-gram retrieval budget; B: verification budget; H: live-tree budget; λ: lookahead threshold

Variables: g: committed output sequence; $\mathcal{T}$: persistent draft tree; $\mathcal{L}_3$: prompt-local n-gram store; A: P1's expansion anchors; R: retrieval trees returned by P0; S: latest published snapshot of $\mathcal{T}$; D: P2's verification subtree; Δg: tokens committed in one verification; Π: target probability observations for updating $\mathcal{L}_3$

P0 loads the available static stores $\mathcal{L}_1, \mathcal{L}_2$ and initializes $\mathcal{L}_3$ from x
In parallel:
  P1: prefill q on x, initialize $\mathcal{T}$, and publish $\mathcal{T}$
  P2: prefill p on x, initialize g with the first target token under ρ, and obtain Π
for all $w \in \{\mathrm{P0}, \mathrm{P1}, \mathrm{P2}\}$ running in parallel **do**
  if $w = \mathrm{P0}$ **then**
    while generation is active **do**
      Serve n-gram queries and update $\mathcal{L}_3$
    end while
  else if $w = \mathrm{P1}$ **then**
    while generation is active **do**
      if P2 requests alignment to g **then**
        $\mathcal{T} \leftarrow \textsc{Align}(\mathcal{T}, g)$; $\textsc{Publish}(\mathcal{T})$
      end if
      if $|\mathcal{T}| < H$ **then**
        $A \leftarrow \textsc{TopKLeaves}(\mathcal{T}, k_a)$
        $R \leftarrow \textsc{Retrieve}(\mathrm{P0}, A, x, g, M)$
        $\mathcal{T} \leftarrow \textsc{Expand}(q, \mathcal{T}, R, k_q)$; $\textsc{Publish}(\mathcal{T})$
      end if
    end while
  else
    while generation is not finished **do**
      Asynchronously update $\mathcal{L}_3$ at P0 with Π
      $S \leftarrow \textsc{LatestSnapshot}(\mathcal{T})$; $S \leftarrow \textsc{Align}(S, g)$; request P1 alignment to g
      $D \leftarrow \textsc{SelectSubtree}(S, B)$
      while $|D| < \min(\lambda, B)$ **do**
        Wait for P1 to publish a new snapshot S; $D \leftarrow \textsc{SelectSubtree}(S, B)$
      end while
      $(\Delta g, \Pi) \leftarrow \textsc{Verify}(p, D, \rho)$; $g \leftarrow g \parallel \Delta g$
    end while
  end if
end for
return the committed output sequence g

Algorithm 2 Caduceus with a fixed batch.

Inputs: prompts $(x_r)_{r=1}^{b}$; models p, q; decoding rule ρ; static stores $\mathcal{L}_1$ and optional $\mathcal{L}_2$
Parameters: batch size b; per-request $k_a, k_q, M, B, H, \lambda$ as in Algorithm 1
Variables: U: unfinished requests; $g_r, \mathcal{T}_r, \mathcal{L}_{3,r}$: per-request output, tree, and local store; A_r, R_r: anchors and retrieval trees; S_r, D_r: aligned snapshot and verification subtree; $\Delta g_r, \Pi_r$: committed tokens and target observations; G, V: P1- and P2-ready rows

$U \leftarrow \{1, \dots, b\}$; P0 loads $\mathcal{L}_1, \mathcal{L}_2$ once and initializes each $\mathcal{L}_{3,r}$ from x_r
In parallel:
  P1: prefill q on all x_r in a padded batch; initialize and publish each $\mathcal{T}_r$
  P2: prefill p on all x_r in a padded batch; initialize each g_r, submit prefill observations to $\mathcal{L}_{3,r}$, and request P1 alignment
for all $w \in \{\mathrm{P0}, \mathrm{P1}, \mathrm{P2}\}$ running in parallel **do**
  if $w = \mathrm{P0}$ **then**
    while $U \neq \emptyset$ **do**
      Serve batched n-gram queries and update each request's $\mathcal{L}_{3,r}$
    end while
  else if $w = \mathrm{P1}$ **then**
    while $U \neq \emptyset$ **do**
      for all $r \in U$ with a pending P2 alignment request **do**
        $\mathcal{T}_r \leftarrow \textsc{Align}(\mathcal{T}_r, g_r)$; $\textsc{Publish}(\mathcal{T}_r)$
      end for
      $G \leftarrow \{r \in U : |\mathcal{T}_r| < H\}$
      if $G \neq \emptyset$ **then**
        $A_r \leftarrow \textsc{TopKLeaves}(\mathcal{T}_r, k_a)$ for each $r \in G$
        $(R_r)_{r \in G} \leftarrow \textsc{RetrieveBatch}(\mathrm{P0}, (A_r, x_r, g_r)_{r \in G}, M)$
        $(\mathcal{T}_r)_{r \in G} \leftarrow \textsc{ExpandBatch}(q, (\mathcal{T}_r, R_r)_{r \in G}, k_q)$
        Publish each updated $\mathcal{T}_r$
      end if
    end while
  else
    while $U \neq \emptyset$ **do**
      $V \leftarrow \emptyset$
      for all $r \in U$ **do**
        $S_r \leftarrow \textsc{LatestSnapshot}(\mathcal{T}_r)$; $S_r \leftarrow \textsc{Align}(S_r, g_r)$
        if $S_r \neq \emptyset$ **then**
          $D_r \leftarrow \textsc{SelectSubtree}(S_r, B)$
          if $|D_r| \geq \min(\lambda, B)$ **then**
            $V \leftarrow V \cup \{r\}$
          end if
        end if
      end for
      if $V = \emptyset$ **then**
        Wait for a new aligned snapshot; **continue**
      end if
      $(\Delta g_r, \Pi_r)_{r \in V} \leftarrow \textsc{VerifyBatch}(p, (D_r)_{r \in V}, \rho)$
      for all $r \in V$ **do**
        $g_r \leftarrow g_r \parallel \Delta g_r$
        Asynchronously submit Π_r to $\mathcal{L}_{3,r}$
        Request P1 alignment to g_r
        if request r has finished **then**
          $U \leftarrow U \setminus \{r\}$
        end if
      end for
    end while
  end if
end for
return $(g_1, \dots, g_b)$

A.1 Ancestor-Only Tree Attention

For both P1 expansion and P2 verification, the tree nodes processed in a model forward pass are linearized into a sequence. Each node uses the position ID determined by its depth from the committed prefix. Let u be a query node, v a key node, and $\mathrm{Anc}(u)$ the strict tree ancestors of u. Attention is computed with an additive logit mask A:

$$\mathrm{Attn}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_k}} + A\right) V.$$

All tree nodes can attend to the committed linear prefix. Within the tree portion, A is

$$A_{uv} = \begin{cases} 0, & v \in \mathrm{Anc}(u) \cup \{u\}, \\ -\infty, & \text{otherwise.} \end{cases} \tag{5}$$

Thus, within the tree, each node attends only to itself and its ancestors, not to siblings or nodes on other branches. Together with depth-consistent position IDs, this mask gives each node the same visible tokens and position as in its corresponding linear sequence. Its logits therefore match those from autoregressive evaluation of that sequence.

A.2 Verification-Subtree Selection Objective

For verification-subtree selection, consider a hypothetical sequence $Y \sim q(\cdot \mid h)$ and let $L(D, Y)$ be its longest prefix represented in D. For a depth-d node v on path $y_{1:d}$, Equation 1 gives

$$w_q(v \mid h) = \exp s_q(v \mid h) = \prod_{i=1}^{d} q(y_i \mid h \,\|\, y_{<i}). \tag{6}$$

At a fixed depth, the events that Y follows different paths are mutually exclusive. Applying the tail-sum identity to the nonnegative integer L yields

$$\mathbb{E}_q[L(D, Y) \mid h] = \sum_{d \geq 1} \Pr_q[L(D, Y) \geq d \mid h] \tag{7}$$

$$= \sum_{d \geq 1} \sum_{v:\, \mathrm{depth}(v) = d} w_q(v \mid h) = \sum_{v \in D} w_q(v \mid h). \tag{8}$$

Consequently, Equation 4 maximizes the expected covered-prefix length under q.

The budgeted selection also has a useful closure property. If u is the parent of a depth-d node v, then

$$w_q(v \mid h) = w_q(u \mid h) q(y_d \mid h \,\|\, y_{<d}) \leq w_q(u \mid h).$$

Thus the B largest node weights, with ancestors ordered before descendants on ties, form an ancestor-closed set and maximize the additive surrogate among all sets of at most B candidate nodes. The implementation therefore sorts nodes by decreasing path score and takes the first B, using node order to place ancestors before descendants on ties.

A.3 N-gram Store Structure and Retrieval

P0 maintains one static count-based tree and one temporary, prompt-local prob-based tree. Table 2 summarizes their structure. Global L1 combines public-corpus L1a with target-specific L1b; optional domain-specific L2 is inserted into the same static tree. L3 is initialized from the prompt tokens and the target-model distributions computed during prefill and is updated with target-model distributions after each verification pass. It is normally discarded after one generation.

Table 2: Structure of the three-level n-gram store. L1a and L1b form global L1; L1 and optional L2 jointly populate the static tree.

Layer	Data source and size	Stored statistics and lifetime
L1a	1,136,344 public-corpus token sequences; source composition is shown in Figure 8	Counts; global and offline; 75% of weighted L1 count mass
L1b	The same 7,374 non-evaluation prompts are used for each target model; every prompt is paired with a response generated by that model. Source composition is shown in Figure 8.	Counts; target-specific and offline; 25% of weighted L1 count mass
L2	Optional domain-matched data	Counts; domain- and target-specific; offline
L3	Current prompt and target-model top-K next-token distributions collected during decoding; no fixed offline size	Averaged probabilities; temporary, prompt-local, and online

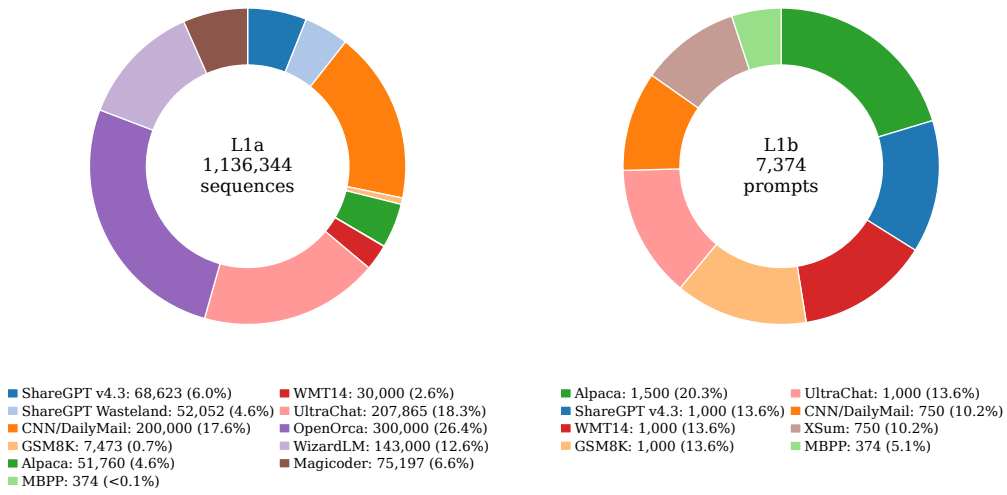


Figure 8: Composition of global L1. L1a sectors represent indexed public-corpus token sequences, whereas L1b sectors represent non-evaluation prompts paired with responses generated by the corresponding target model. The panels use independent totals; Table 2 reports their 75%/25% weighted count-mass mixture.

Our implementation of both stores builds on and extends the released SuffixDecoding codebase (Oliaro et al., 2025). Given a query history, each store finds its longest matching suffix and constructs a candidate tree by best-first expansion. Available nodes are ranked by the product of the edge weights in Equations 2 and 3 along their root-to-node paths; the highest-scoring node is added and its children become candidates until M nodes have been selected or no candidates remain. P0 then merges the two trees by root-to-node token prefix. A unique edge retains its source weight, while a shared edge keeps $r_{\cup}(e) = \max\{r_{\text{stat}}(e), r_{\text{dyn}}(e)\}$. Starting from the root, P0 assigns

$$R(\text{root}) = 1, \qquad R(v) = R(\text{pa}(v))\, r_{\cup}(\text{pa}(v){\to}v), \tag{9}$$

where $\text{pa}(v)$ is the parent of v. If the union exceeds M nodes, P0 considers nodes by decreasing $R(v)$ and retains each only with its missing ancestors. The resulting retrieval tree is ancestor-closed and contains at most M nodes.

A.4 Pipeline Critical-Path Model

We use a target-side decomposition to identify which operations remain on P2's critical path and how local snapshot alignment reduces target idle time. Because P1 grows $\mathcal{T}$ continuously, its work does not form one-to-one rounds with P2. Let T_t^{pipe} be the elapsed time between completion of target verification passes $t-1$ and t, and let Θ_J^{dec} denote decoding-stage throughput over J passes:

$$T_t^{\text{pipe}} = v_t + o_t + w_t, \qquad \Theta_J^{\text{dec}} = \frac{\sum_{t=1}^{J} \tau_t}{\sum_{t=1}^{J} T_t^{\text{pipe}}}, \tag{10}$$

Here v_t is the target-model tree-forward time, and τ_t is the number of tokens committed by pass t. The term o_t contains the remaining P2 work on the path to its next verification: verification-buffer construction and transfer, posterior and acceptance processing, committed-path update and boundary-token selection, KV-cache compaction, local snapshot alignment, verification-subtree selection, alignment-request coordination, and L3-observation submission. The L3 update itself runs asynchronously.

The term w_t is the residual idle time after P2 has completed this work but no aligned snapshot satisfying the $\min(\lambda, B)$ node threshold is available. The execution time of P0 and P1 therefore enters the target-side critical path directly only through w_t. If local alignment of the latest snapshot succeeds with enough nodes, P2 proceeds to its next verification without waiting for P1 and $w_t = 0$; otherwise, it waits for P1's authoritative alignment and a sufficiently large snapshot. This decomposition makes the pipeline objective explicit: keep P0 and P1 off the target-side critical path by minimizing w_t. It also exposes a limitation: a substantial part of o_t is introduced by the Caduceus pipeline and is less easily amortized when v_t is small, which can limit speedup for smaller target models such as the 8B model.

B Additional Experimental Results

B.1 Detailed Spec-Bench and HumanEval Results

Tables 3 and 4 report the absolute throughput and speedups underlying the main figures. Table 5 reports committed tokens per target verification pass for the four Caduceus variants. Table entries use "–" only for unavailable model configurations.

Table 3: Spec-Bench throughput and speedup over target-only AR. TPS is the arithmetic mean of per-prompt throughput.

	Llama 1B→8B		Qwen 7B→32B		Vicuna 7B→33B		Llama 1B→70B	
Method	TPS	Speedup	TPS	Speedup	TPS	Speedup	TPS	Speedup
Target AR	28.48	1.00×	9.89	1.00×	11.20	1.00×	6.26	1.00×
PLD	36.16	1.27×	11.86	1.20×	15.09	1.35×	7.68	1.23×
SuffixDecoding	41.09	1.44×	13.08	1.32×	16.01	1.43×	8.05	1.29×
Token Recycling	51.96	1.82×	12.39	1.25×	13.58	1.21×	6.97	1.11×
HF Assisted Gen.	25.60	0.90×	12.60	1.27×	12.90	1.15×	12.38	1.98×
PipeSpec async	30.71	1.08×	13.13	1.33×	14.84	1.32×	10.89	1.74×
CS-Drafting	33.31	1.17×	14.73	1.49×	16.59	1.48×	13.01	2.08×
PEARL	37.31	1.31×	16.88	1.71×	15.99	1.43×	16.22	2.59×
EAGLE-2	46.06	1.62×	–	–	25.86	2.31×	–	–
EAGLE-3	59.29	2.08×	–	–	–	–	18.07	2.89×
Caduceus-Ngram	44.62	1.57×	15.89	1.61×	19.01	1.70×	9.99	1.60×
Caduceus-SM	36.72	1.29×	19.66	1.99×	20.12	1.80×	18.72	2.99×
Caduceus-Serial	49.98	1.75×	20.69	2.09×	22.12	1.97×	17.31	2.77×
Caduceus	60.76	2.13×	30.16	3.05×	31.13	2.78×	23.16	3.70×

Table 4: HumanEval throughput and speedup over target-only AR. TPS is the arithmetic mean of per-prompt throughput.

	Llama 1B→8B		Qwen 7B→32B		Vicuna 7B→33B		Llama 1B→70B	
Method	TPS	Speedup	TPS	Speedup	TPS	Speedup	TPS	Speedup
Target AR	28.18	1.00×	9.62	1.00×	11.05	1.00×	6.05	1.00×
PLD	29.09	1.03×	10.59	1.10×	14.99	1.36×	6.66	1.10×
SuffixDecoding	43.01	1.53×	13.68	1.42×	19.74	1.79×	7.57	1.25×
Token Recycling	46.40	1.65×	11.62	1.21×	13.56	1.23×	6.56	1.08×
HF Assisted Gen.	26.89	0.95×	14.21	1.48×	12.72	1.15×	12.82	2.12×
PipeSpec async	32.56	1.16×	13.08	1.36×	15.33	1.39×	11.62	1.92×
CS-Drafting	33.00	1.17×	15.31	1.59×	17.60	1.59×	14.25	2.35×
PEARL	38.72	1.37×	20.69	2.15×	15.87	1.44×	18.04	2.98×
EAGLE-2	46.14	1.64×	–	–	30.14	2.73×	–	–
EAGLE-3	55.53	1.97×	–	–	–	–	17.18	2.84×
Caduceus-Ngram	53.43	1.90×	18.66	1.94×	22.47	2.03×	10.78	1.78×
Caduceus-SM	36.32	1.29×	19.92	2.07×	19.86	1.80×	17.92	2.96×
Caduceus-Serial	46.71	1.66×	22.01	2.29×	23.94	2.17×	16.63	2.75×
Caduceus	64.06	2.27×	31.46	3.27×	33.18	3.00×	23.39	3.86×

Table 5: Mean output tokens committed per target verification pass (τ) for the four Caduceus variants. Values are first averaged within each request and then across requests.

	Spec-Bench				HumanEval			
Method	8B	32B	33B	70B	8B	32B	33B	70B
Caduceus-Ngram	2.417	1.954	2.105	1.999	2.803	2.316	2.589	2.241
Caduceus-SM	2.003	2.639	2.420	4.274	1.987	2.604	2.441	4.129
Caduceus-Serial	5.912	7.848	7.312	8.035	9.325	9.062	8.563	8.113
Caduceus	3.639	4.295	4.050	5.318	3.817	4.770	4.466	5.646

Table 6 reports the exact speedups in Figure 6 together with the corresponding τ values.

Table 6: Exact speedups and committed tokens per target verification pass (τ) for the L1/L3 complementarity ablation in Figure 6.

		Small model only		+L1		+L3		+L1 & L3	
Model pair	Benchmark	Speedup	τ	Speedup	τ	Speedup	τ	Speedup	τ
Llama 1B→8B	Spec-Bench	1.29×	2.00	1.69×	2.70	1.98×	3.35	2.13×	3.64
Llama 1B→8B	HumanEval	1.29×	1.99	2.02×	3.20	1.73×	2.82	2.27×	3.82
Qwen 7B→32B	Spec-Bench	1.99×	2.64	2.56×	3.49	2.69×	3.89	3.05×	4.29
Qwen 7B→32B	HumanEval	2.07×	2.60	3.09×	4.28	2.53×	3.43	3.27×	4.77
Vicuna 7B→33B	Spec-Bench	1.80×	2.42	2.29×	3.29	2.52×	3.79	2.78×	4.05
Vicuna 7B→33B	HumanEval	1.80×	2.44	2.53×	3.66	2.54×	3.82	3.00×	4.47
Llama 1B→70B	Spec-Bench	2.99×	4.27	3.51×	4.97	3.44×	5.08	3.70×	5.32
Llama 1B→70B	HumanEval	2.96×	4.13	3.71×	5.46	3.41×	4.73	3.86×	5.65

Tables 7–10 report the complete domain-level results; Figure 5 visualizes a representative subset.

Table 7: Spec-Bench domain-level throughput and speedup for Llama 1B→8B. Each entry reports TPS followed by speedup over target-only AR in parentheses; unavailable model-specific configurations are marked –.

Method	MT Chat	Translation	Summ.	QA	Math	RAG	Overall
Target AR	28.74 (1.00×)	28.47 (1.00×)	28.61 (1.00×)	28.74 (1.00×)	28.71 (1.00×)	27.63 (1.00×)	28.48 (1.00×)
PLD	31.05 (1.08×)	34.84 (1.22×)	40.00 (1.40×)	34.66 (1.21×)	34.28 (1.19×)	42.16 (1.53×)	36.16 (1.27×)
SuffixDecoding	36.25 (1.26×)	38.75 (1.36×)	40.04 (1.40×)	39.57 (1.38×)	44.53 (1.55×)	47.39 (1.71×)	41.09 (1.44×)
Token Recycling	55.80 (1.94×)	41.93 (1.47×)	54.54 (1.91×)	50.01 (1.74×)	62.57 (2.18×)	46.93 (1.70×)	51.96 (1.82×)
HF Assisted Gen.	26.27 (0.91×)	24.34 (0.85×)	24.73 (0.86×)	26.03 (0.91×)	27.17 (0.95×)	25.07 (0.91×)	25.60 (0.90×)
PipeSpec async	33.31 (1.16×)	39.22 (1.38×)	25.28 (0.88×)	29.66 (1.03×)	31.95 (1.11×)	24.83 (0.90×)	30.71 (1.08×)
CS-Drafting	33.58 (1.17×)	29.08 (1.02×)	33.31 (1.16×)	32.89 (1.14×)	37.48 (1.31×)	33.50 (1.21×)	33.31 (1.17×)
PEARL	38.39 (1.34×)	36.20 (1.27×)	36.36 (1.27×)	37.98 (1.32×)	40.80 (1.42×)	34.15 (1.24×)	37.31 (1.31×)
EAGLE-2	52.70 (1.83×)	34.70 (1.22×)	44.41 (1.55×)	43.32 (1.51×)	56.58 (1.97×)	44.65 (1.62×)	46.06 (1.62×)
EAGLE-3	70.73 (2.46×)	35.65 (1.25×)	63.25 (2.21×)	54.27 (1.89×)	73.60 (2.56×)	58.24 (2.11×)	59.29 (2.08×)
Caduceus-Ngram	49.84 (1.73×)	34.83 (1.22×)	35.76 (1.25×)	50.92 (1.77×)	58.29 (2.03×)	38.06 (1.38×)	44.62 (1.57×)
Caduceus-SM	36.76 (1.28×)	34.58 (1.21×)	36.56 (1.28×)	37.02 (1.29×)	38.99 (1.36×)	36.40 (1.32×)	36.72 (1.29×)
Caduceus-Serial	51.12 (1.78×)	41.80 (1.47×)	54.95 (1.92×)	48.42 (1.69×)	55.61 (1.94×)	47.98 (1.74×)	49.98 (1.75×)
Caduceus	63.04 (2.19×)	52.42 (1.84×)	60.07 (2.10×)	59.90 (2.08×)	71.03 (2.47×)	58.10 (2.10×)	60.76 (2.13×)

Table 8: Spec-Bench domain-level throughput and speedup for Qwen 7B→32B. Each entry reports TPS followed by speedup over target-only AR in parentheses; unavailable model-specific configurations are marked –.

Method	MT Chat	Translation	Summ.	QA	Math	RAG	Overall
Target AR	9.95 (1.00×)	10.12 (1.00×)	9.71 (1.00×)	10.18 (1.00×)	10.11 (1.00×)	9.28 (1.00×)	9.89 (1.00×)
PLD	11.65 (1.17×)	10.20 (1.01×)	12.97 (1.33×)	10.32 (1.01×)	14.30 (1.41×)	11.70 (1.26×)	11.86 (1.20×)
SuffixDecoding	12.95 (1.30×)	10.86 (1.07×)	13.05 (1.34×)	11.73 (1.15×)	17.44 (1.73×)	12.45 (1.34×)	13.08 (1.32×)
Token Recycling	13.46 (1.35×)	8.62 (0.85×)	12.21 (1.26×)	12.18 (1.20×)	16.71 (1.65×)	11.13 (1.20×)	12.39 (1.25×)
HF Assisted Gen.	12.99 (1.31×)	13.39 (1.32×)	11.49 (1.18×)	10.96 (1.08×)	16.09 (1.59×)	10.69 (1.15×)	12.60 (1.27×)
PipeSpec async	15.34 (1.54×)	10.93 (1.08×)	10.08 (1.04×)	12.57 (1.23×)	18.32 (1.81×)	11.54 (1.24×)	13.13 (1.33×)
CS-Drafting	15.25 (1.53×)	14.80 (1.46×)	13.65 (1.41×)	12.13 (1.19×)	20.14 (1.99×)	12.38 (1.33×)	14.73 (1.49×)
PEARL	17.26 (1.73×)	18.89 (1.87×)	14.40 (1.48×)	14.34 (1.41×)	23.00 (2.28×)	13.37 (1.44×)	16.88 (1.71×)
EAGLE-2	–	–	–	–	–	–	–
EAGLE-3	–	–	–	–	–	–	–
Caduceus-Ngram	17.25 (1.73×)	12.37 (1.22×)	13.31 (1.37×)	15.90 (1.56×)	22.84 (2.26×)	13.65 (1.47×)	15.89 (1.61×)
Caduceus-SM	20.83 (2.09×)	19.29 (1.91×)	18.67 (1.92×)	18.33 (1.80×)	23.16 (2.29×)	17.69 (1.91×)	19.66 (1.99×)
Caduceus-Serial	23.48 (2.36×)	14.37 (1.42×)	19.42 (2.00×)	17.86 (1.75×)	31.47 (3.11×)	17.54 (1.89×)	20.69 (2.09×)
Caduceus	33.22 (3.34×)	24.57 (2.43×)	26.88 (2.77×)	24.74 (2.43×)	47.43 (4.69×)	24.13 (2.60×)	30.16 (3.05×)

Table 9: Spec-Bench domain-level throughput and speedup for Vicuna 7B→33B. Each entry reports TPS followed by speedup over target-only AR in parentheses; unavailable model-specific configurations are marked –.

Method	MT Chat	Translation	Summ.	QA	Math	RAG	Overall
Target AR	11.40 (1.00×)	11.40 (1.00×)	11.09 (1.00×)	11.59 (1.00×)	11.60 (1.00×)	10.14 (1.00×)	11.20 (1.00×)
PLD	14.70 (1.29×)	11.48 (1.01×)	21.53 (1.94×)	12.08 (1.04×)	17.33 (1.49×)	13.44 (1.33×)	15.09 (1.35×)
SuffixDecoding	16.26 (1.43×)	11.91 (1.04×)	19.55 (1.76×)	14.23 (1.23×)	20.31 (1.75×)	13.79 (1.36×)	16.01 (1.43×)
Token Recycling	14.22 (1.25×)	12.00 (1.05×)	13.29 (1.20×)	13.14 (1.13×)	17.35 (1.49×)	11.48 (1.13×)	13.58 (1.21×)
HF Assisted Gen.	12.91 (1.13×)	14.74 (1.29×)	13.00 (1.17×)	11.76 (1.01×)	13.79 (1.19×)	11.18 (1.10×)	12.90 (1.15×)
PipeSpec async	16.40 (1.44×)	12.35 (1.08×)	13.71 (1.24×)	15.28 (1.32×)	17.56 (1.51×)	13.73 (1.35×)	14.84 (1.32×)
CS-Drafting	16.73 (1.47×)	16.29 (1.43×)	20.18 (1.82×)	13.39 (1.16×)	19.11 (1.65×)	13.84 (1.37×)	16.59 (1.48×)
PEARL	16.44 (1.44×)	17.20 (1.51×)	15.79 (1.42×)	16.14 (1.39×)	17.37 (1.50×)	13.00 (1.28×)	15.99 (1.43×)
EAGLE-2	29.38 (2.58×)	21.24 (1.86×)	25.30 (2.28×)	24.81 (2.14×)	34.16 (2.94×)	20.24 (2.00×)	25.86 (2.31×)
EAGLE-3	–	–	–	–	–	–	–
Caduceus-Ngram	21.65 (1.90×)	14.83 (1.30×)	16.41 (1.48×)	20.22 (1.75×)	25.49 (2.20×)	15.44 (1.52×)	19.01 (1.70×)
Caduceus-SM	20.50 (1.80×)	20.84 (1.83×)	20.46 (1.85×)	19.92 (1.72×)	21.56 (1.86×)	17.42 (1.72×)	20.12 (1.80×)
Caduceus-Serial	24.34 (2.14×)	16.94 (1.49×)	25.67 (2.31×)	20.29 (1.75×)	27.29 (2.35×)	18.17 (1.79×)	22.12 (1.97×)
Caduceus	34.29 (3.01×)	26.62 (2.34×)	35.24 (3.18×)	28.51 (2.46×)	37.95 (3.27×)	24.17 (2.38×)	31.13 (2.78×)

Table 10: Spec-Bench domain-level throughput and speedup for Llama 1B→70B. Each entry reports TPS followed by speedup over target-only AR in parentheses; unavailable model-specific configurations are marked –.

Method	MT Chat	Translation	Summ.	QA	Math	RAG	Overall
Target AR	6.47 (1.00×)	6.46 (1.00×)	6.26 (1.00×)	6.55 (1.00×)	6.46 (1.00×)	5.33 (1.00×)	6.26 (1.00×)
PLD	7.69 (1.19×)	6.29 (0.97×)	8.69 (1.39×)	7.42 (1.13×)	8.89 (1.38×)	7.12 (1.34×)	7.68 (1.23×)
SuffixDecoding	8.14 (1.26×)	6.91 (1.07×)	8.77 (1.40×)	7.70 (1.18×)	9.39 (1.45×)	7.37 (1.38×)	8.05 (1.29×)
Token Recycling	7.47 (1.16×)	5.45 (0.84×)	7.00 (1.12×)	7.26 (1.11×)	8.92 (1.38×)	5.70 (1.07×)	6.97 (1.11×)
HF Assisted Gen.	12.95 (2.00×)	10.18 (1.58×)	10.41 (1.66×)	12.87 (1.96×)	18.74 (2.90×)	9.13 (1.71×)	12.38 (1.98×)
PipeSpec async	13.04 (2.02×)	7.43 (1.15×)	8.46 (1.35×)	11.99 (1.83×)	17.27 (2.67×)	7.14 (1.34×)	10.89 (1.74×)
CS-Drafting	13.18 (2.04×)	9.70 (1.50×)	11.20 (1.79×)	12.90 (1.97×)	21.53 (3.33×)	9.56 (1.79×)	13.01 (2.08×)
PEARL	16.62 (2.57×)	13.62 (2.11×)	12.91 (2.06×)	19.26 (2.94×)	25.53 (3.95×)	9.39 (1.76×)	16.22 (2.59×)
EAGLE-2	–	–	–	–	–	–	–
EAGLE-3	21.52 (3.33×)	15.10 (2.34×)	18.00 (2.87×)	17.80 (2.72×)	22.92 (3.55×)	13.10 (2.46×)	18.07 (2.89×)
Caduceus-Ngram	11.20 (1.73×)	8.41 (1.30×)	8.83 (1.41×)	10.26 (1.57×)	13.80 (2.14×)	7.44 (1.39×)	9.99 (1.60×)
Caduceus-SM	20.32 (3.14×)	15.85 (2.45×)	16.55 (2.64×)	18.48 (2.82×)	28.38 (4.39×)	12.75 (2.39×)	18.72 (2.99×)
Caduceus-Serial	18.70 (2.89×)	13.32 (2.06×)	16.07 (2.57×)	16.49 (2.52×)	27.48 (4.25×)	11.80 (2.21×)	17.31 (2.77×)
Caduceus	24.94 (3.86×)	17.75 (2.75×)	19.34 (3.09×)	21.68 (3.31×)	40.91 (6.33×)	14.32 (2.69×)	23.16 (3.70×)

B.2 Detailed SpecTTS-Bench Results

Tables 11 and 12 report the absolute throughput and speedups for the rollout-level results in Figure 7 and the four problem categories, respectively.

Table 11: SpecTTS-Bench rollout-level throughput. TPS is the arithmetic mean of per-prompt throughput; parenthesized speedups use the matching target-only AR mean. Each rollout contains 120 prompts.

Target	Rollout	AR TPS	EAGLE-3	Caduceus-Ngram	Caduceus
8B	1	30.98	60.83 (1.96×)	58.73 (1.90×)	63.94 (2.06×)
8B	2	30.85	61.37 (1.99×)	69.57 (2.26×)	69.19 (2.24×)
8B	3	30.86	60.80 (1.97×)	74.68 (2.42×)	71.50 (2.32×)
8B	4	30.91	60.75 (1.97×)	78.13 (2.53×)	73.88 (2.39×)
70B	1	7.11	20.54 (2.89×)	12.13 (1.71×)	28.39 (3.99×)
70B	2	7.10	20.31 (2.86×)	15.88 (2.24×)	29.63 (4.18×)
70B	3	7.10	20.24 (2.85×)	17.51 (2.47×)	31.18 (4.39×)
70B	4	7.11	20.60 (2.90×)	19.15 (2.69×)	30.50 (4.29×)

Table 12: SpecTTS-Bench domain-level throughput. TPS is the arithmetic mean of per-prompt throughput; parenthesized speedups use the matching target-only AR mean. Each category contains 30 prompts evaluated over four rollouts.

Target	Domain	AR TPS	EAGLE-3	Caduceus-Ngram	Caduceus
8B	AIME24	31.31	58.56 (1.87×)	67.20 (2.15×)	67.44 (2.15×)
8B	AIME25	30.95	59.80 (1.93×)	66.79 (2.16×)	69.60 (2.25×)
8B	MATH-500	30.90	65.10 (2.11×)	81.66 (2.64×)	78.42 (2.54×)
8B	GPQA	30.44	60.29 (1.98×)	65.45 (2.15×)	63.04 (2.07×)
8B	Overall	30.90	60.94 (1.97×)	70.28 (2.27×)	69.63 (2.25×)
70B	AIME24	7.08	20.48 (2.89×)	15.68 (2.21×)	30.33 (4.28×)
70B	AIME25	7.10	20.42 (2.88×)	15.58 (2.19×)	28.60 (4.03×)
70B	MATH-500	7.13	22.44 (3.15×)	18.95 (2.66×)	36.69 (5.15×)
70B	GPQA	7.12	18.35 (2.58×)	14.46 (2.03×)	24.08 (3.38×)
70B	Overall	7.11	20.42 (2.87×)	16.17 (2.28×)	29.93 (4.21×)

B.3 Effect of Domain-Specific L2 Retrieval

L2 is an optional static store built from text matched to the deployment domain. To isolate its marginal value, we compare the default L1+L3 configuration with L1+L2+L3 while keeping the small model, target model, tree budgets, and decoding protocol fixed. We report throughput and committed tokens per target verification pass because broad benchmark averages can dilute domain-specific gains.

Table 13 evaluates this effect on held-out rare-disease cases. We deterministically split RareArena into 1,600 cases for constructing L2 and 500 disjoint cases for evaluation. For each target model, we generate responses for the L2 split and index the resulting prompt–response sequences in a target-specific store. L1 and L2 are merged into one static count-based tree with an approximate 3:1 L1:L2 token-mass ratio.

Table 13: Effect of domain-specific L2 retrieval on held-out RareArena rare-disease cases. TPS and τ are means over 500 examples. Both conditions keep L3 enabled and use the same model pair, tree budget, and decoding protocol; only the static datastore changes from L1 to merged L1+L2.

Draft→target	L1 TPS	L1+L2 TPS	Relative gain	L1 τ	L1+L2 τ
Llama 1B→8B	50.424	59.048	1.171×	3.261	3.947
Qwen 7B→32B	24.360	26.911	1.105×	3.667	4.002
Vicuna 7B→33B	29.642	32.872	1.109×	3.969	4.486
Llama 1B→70B	16.682	17.836	1.069×	3.803	4.031

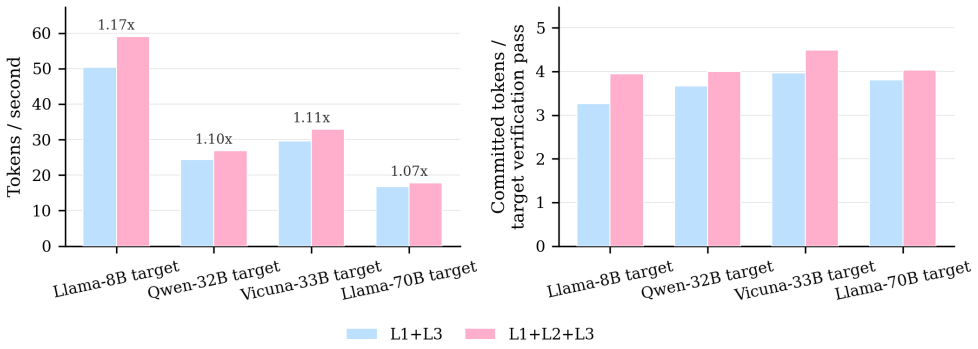


Figure 9: Effect of domain-specific L2 retrieval on held-out RareArena cases. Each pair compares L1+L3 with L1+L2+L3 under the same model pair and decoding budget. Adding L2 increases mean throughput and committed tokens per target verification pass across all four settings.

As visualized in Figure 9, adding L2 increases mean throughput by 6.9–17.1% across the four model pairs and also increases τ.

B.4 Lookahead Sensitivity

Figures 10–12 compare two ways of increasing lookahead. For Caduceus, we vary the lookahead threshold $\lambda \in \{0, 1, 2, 4, 8, 16, 32, 64\}$ on all 480 Spec-Bench prompts for the four model pairs. For Caduceus-Serial, we instead vary the number of compound-drafting iterations over $\{1, 2, 4, 6\}$ before each target verification pass on Spec-Bench and HumanEval.

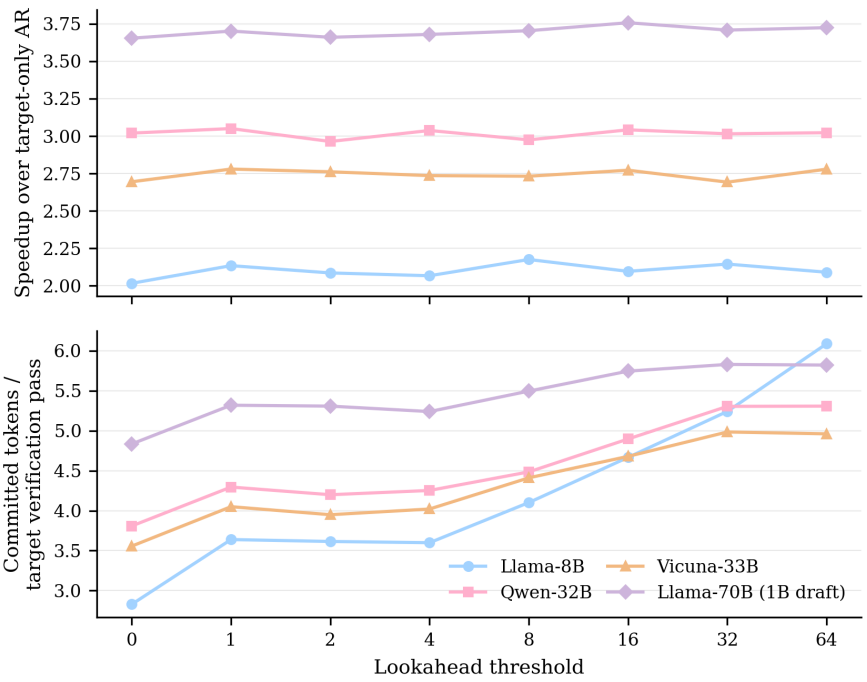


Figure 10: Caduceus speedup and τ across lookahead thresholds λ on Spec-Bench.

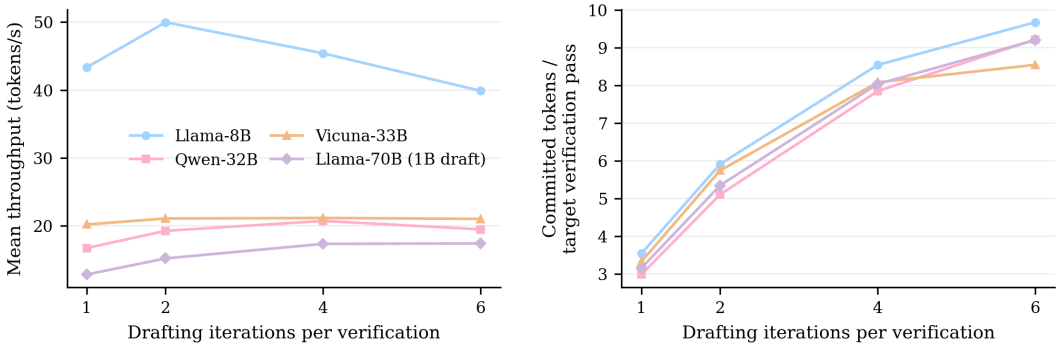


Figure 11: Caduceus-Serial throughput and τ across numbers of drafting iterations on Spec-Bench.

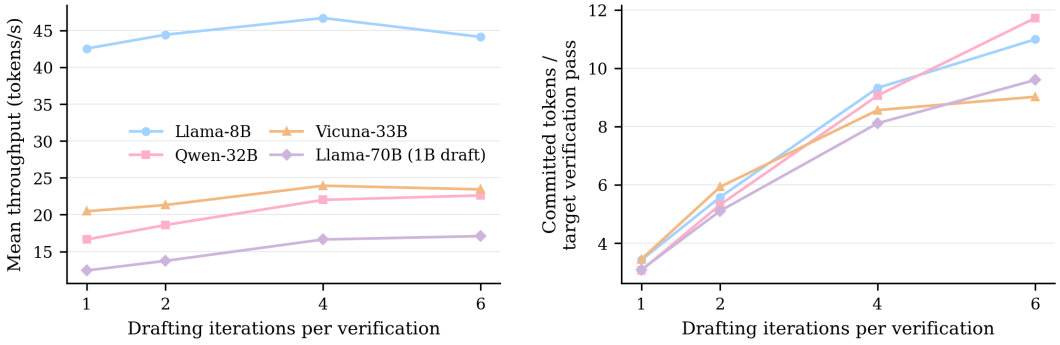


Figure 12: Caduceus-Serial throughput and τ across numbers of drafting iterations on HumanEval.

These results show that Caduceus is largely insensitive to the lookahead threshold λ: its speedup remains stable even as τ changes. In contrast, Caduceus-Serial is sensitive to lookahead depth and exhibits a configuration-dependent throughput optimum.

C Cost Analysis

C.1 Target-Model Tree-Forward Cost

Figure 13 isolates how target-forward latency scales with the number of input draft tokens when the prefix KV cache is already populated. Llama-8B remains near its latency plateau through 256 tokens and rises at 512. Qwen-32B remains nearly flat through 32 tokens, begins rising at 64, and is close to twice its one-token latency by 128. Vicuna-33B also turns upward at 64 and exceeds $2\times$ by 128. Llama-70B follows the same earlier knee, reaching 2.37–$2.60\times$ at 128 tokens. Thus, the smaller 8B target has a substantially broader low-cost region than the larger targets, motivating $B = 64$ for Llama-8B and smaller budgets for Qwen-32B, Vicuna-33B, and Llama-70B.

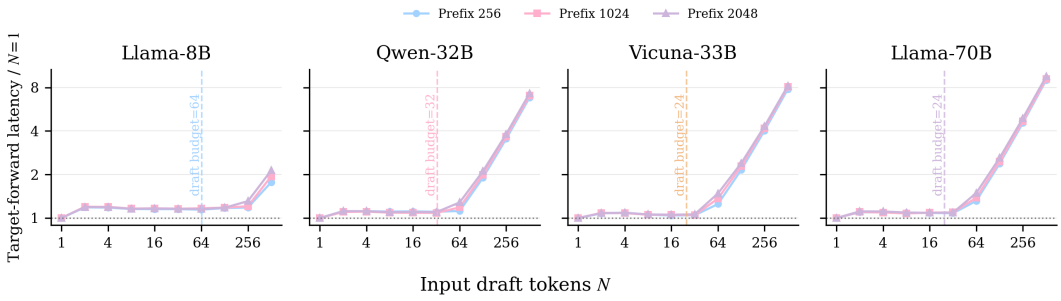


Figure 13: Target-model tree-forward latency normalized by the corresponding one-token forward. Dashed lines mark the verification budgets used for the four target models.

C.2 Draft-Model Tree-Forward Cost

Figure 14 measures the P1 tree-forward path for the three off-the-shelf draft models. Prefix length has little effect over the measured range. Llama-1B remains nearly flat from 2 to 512 tree tokens, while Qwen-7B and Vicuna-7B remain nearly flat through 256 tokens before their latency rises at 512. These plateaus allow P1 to process multi-branch retrieval trees without making its forward cost proportional to the number of tree nodes over the operating range.

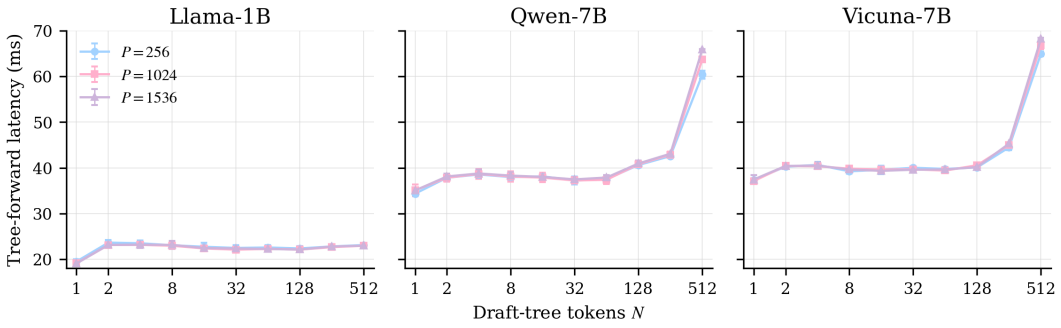


Figure 14: Draft-model latency for one P1 tree forward over N tree tokens at prefix length P.

C.3 Batched Tree-Forward Cost

Figure 15 isolates model-forward cost under the batching path used for the Llama 1B→70B configuration. The 1B drafter remains in a broad latency plateau across the tested batch sizes. For the 70B target, the low-cost range narrows as batch size increases: latency begins to rise at progressively smaller per-request tree sizes N. These knees correspond to roughly 32–64 total tree-token rows per forward and motivate reducing the per-request verification budget as batch size grows.

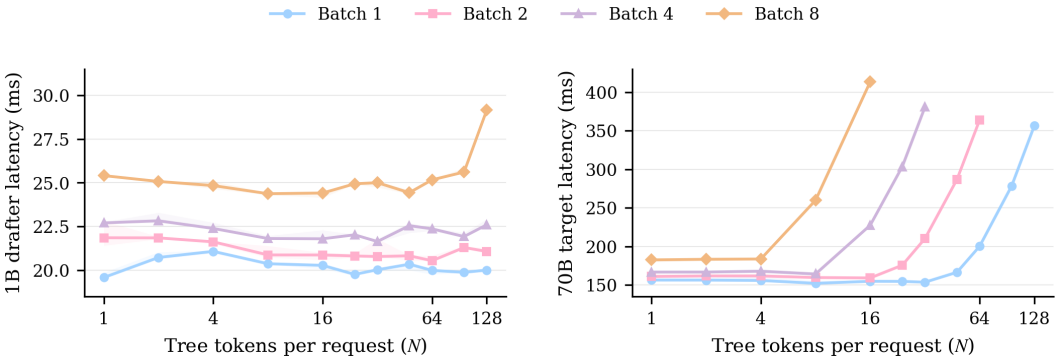


Figure 15: Batched tree-forward latency versus per-request tree size N for the Llama-1B drafter and Llama-70B target with a 1,024-token cached prefix.

D Case Study

We trace one Spec-Bench Math request with the Llama-70B target. The prompt asks: *Mary buys 3 bags of M&Ms. The first bag has 300 M&Ms in it. The second bag has 12 more M&Ms than the first, and the third bag has a hole in it, so it only has half the number of M&Ms that the first bag had. How many M&Ms did Mary get total?* Both methods use greedy decoding with a 512-token output cap. Model loading is outside the timed interval, while prompt prefill is inside it. Caduceus uses the Llama-1B drafter and L1+L3 configuration, and EAGLE-3 uses the target-specific trained head and the Spec-Bench parameters of the main comparison. The 70B target occupies two GPUs in both runs; Caduceus places the 1B drafter on another GPU, whereas the EAGLE-3 head shares the GPU holding the target's last layer.

Figure 16 records the measured Caduceus run. The two lanes mark prefill (P), n-gram retrieval (N, P1 only), model forward (F), and alignment (A). The lower trace gives the number of nodes in each published shared-memory tree snapshot. Its rises are draft growth; drops follow alignment with a committed target path.

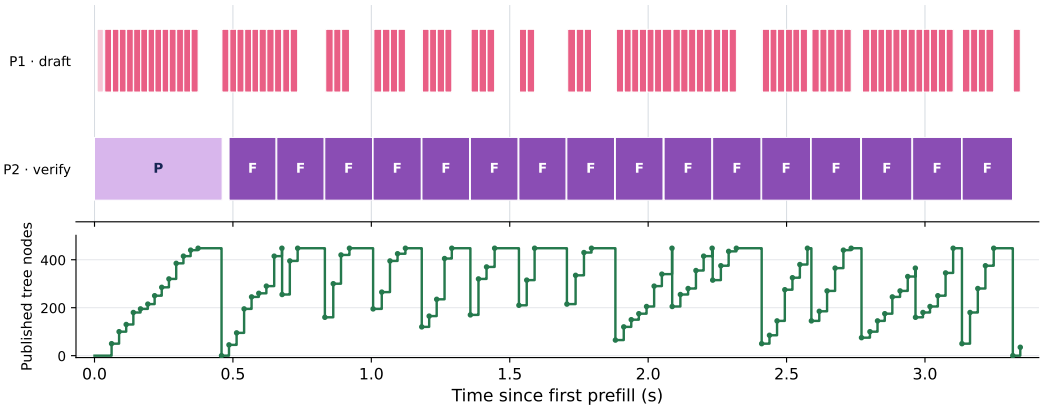


Figure 16: Caduceus pipeline for one Spec-Bench Math prompt. The lower curve shows the size of each published tree snapshot.

On this prompt, target-only AR achieved 6.49 tokens/s. Caduceus generated 189 tokens in 3.32 s (56.93 tokens/s; 8.77× AR) using 16 target verification passes. EAGLE-3 generated 187 tokens in 6.63 s (28.22 tokens/s; 4.35× AR) using 30 passes. Thus, Caduceus delivered 2.02× EAGLE-3's throughput on this request. Table 14 shows both outputs as text. Red marks the second and subsequent tokens committed by each target-model forward, i.e., the additional tokens obtained from draft verification beyond the single token produced by AR decoding.

Table 14: Generated outputs for the same Spec-Bench Math prompt. Red text marks tokens committed beyond the first token in each target verification pass.

Method	Generated output
Caduceus 8.77×	## Step 1: Calculate the number of M&Ms in the first bag. The first bag has 300 M&Ms. ## Step 2: Calculate the number of M&Ms in the second bag. The second bag has 12 more M&Ms than the first bag, so it has 300 + 12 = 312 M&Ms. ## Step 3: Calculate the number of M&Ms in the third bag. The third bag has half the number of M&Ms that the first bag had, so it has 300 / 2 = 150 M&Ms. ## Step 4: Calculate the total number of M&Ms Mary got. To find the total number of M&Ms, we need to add the number of M&Ms in each bag: 300 + 312 + 150 = 762 M&Ms. The final answer is: $\boxed{762}$
EAGLE-3 4.35×	## Step 1: Calculate the number of M&Ms in the first bag. The first bag has 300 M&Ms. ## Step 2: Calculate the number of M&Ms in the second bag. The second bag has 12 more M&Ms than the first bag, so it has 300 + 12 = 312 M&Ms. ## Step 3: Calculate the number of M&Ms in the third bag. The third bag has half the number of M&Ms that the first bag had, so it has 300 / 2 = 150 M&Ms. ## Step 4: Calculate the total number of M&Ms Mary got. To find the total number of M&Ms, we add the number of M&Ms in each bag: 300 + 312 + 150 = 762 M&Ms. The final answer is: $\boxed{762}$